Exercise walk-through

Computational Thinking Skills ▪ 9.1

eXercise

You must be able to

- explain and use **abstraction** 抽象
- explain and use **decomposition** 分解

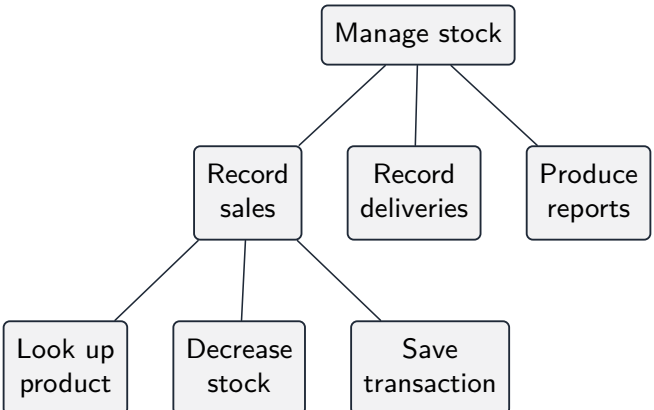
Method — Abstraction

Abstraction means keeping the **essential** features of a problem and **ignoring irrelevant detail**, to give a simpler model. Examples: a train-line map keeps the stations and lines but drops the real geography; a function hides a job behind a name; a class keeps only the attributes the system needs. Without abstraction, a full model of a real problem would be too big to reason about.

Method — Decomposition

Decomposition means breaking a large problem into smaller **sub-problems**, each easier to solve and combined at the end:
It makes a big task manageable, lets a **team** divide the work, and produces **modular** code that is easier to test and maintain.

Method — Decomposition



Method — Decomposition

real geography

abstraction

metro map

keep stations + lines, drop the geography

Abstraction hides detail to manage complexity

Method — Pattern recognition

Pattern recognition 模式识别 means spotting where parts of a problem are the **same or similar**, so one solution can handle them all — for example, noticing that several jobs repeat lets you use a **loop** or reuse one **module** instead of writing the same code many times.

In the exam you may be asked to **identify which skill** a designer used: keeping only the needed detail → **abstraction**; splitting the problem into parts → **decomposition**; spotting repetition → **pattern recognition**.

Practice 2.1 ▪ 1 mark

State what **abstraction** means.

Solution 2.1

Method: Abstraction

Worked steps

Keeping the **essential features** of a problem and **ignoring irrelevant detail** (a simpler model) B1.

Practice 2.2 ▪ 1 mark

State what **decomposition** means.

Solution 2.2

Worked steps

Breaking a large problem into smaller, easier **sub-problems** B1.

Practice 2.3 ▪ 1 mark

Give one benefit of decomposing a problem.

Solution 2.3

Method: Abstraction

Worked steps

Any one: makes the problem manageable; lets a team share the work; gives modular, reusable, easier-to-test code B1.

Practice 2.4 ▪ 1 mark

Give one everyday example of abstraction.

Solution 2.4

Worked steps

Any one: a map; a function/method; a model/diagram that drops detail B1.

Exam-style 3.1 ▪ 3 marks

Explain what is meant by **abstraction**, giving an example.

Solution 3.1

Worked steps

Abstraction keeps the **essential** features and removes irrelevant detail B1, giving a simpler model that is easier to work with B1; example, e.g. a metro map shows lines and stops but not real distances B1.

Exam-style 3.2 ▪ 3 marks

A team must build a system to run a school library (lending and returning books, managing members, fines). Use **decomposition** to break this into **three** sub-tasks.

Solution 3.2

Worked steps

Any three sensible sub-tasks, e.g.: lend/return books; manage member records; calculate and record fines; search the catalogue B1 B1 B1.

Exam-style 3.3 ▪ 2 marks

Explain **two** benefits of decomposing a large program into modules.

Solution 3.3

Worked steps

Any two: smaller parts are easier to solve/test; a team can work in parallel; modules can be reused; maintenance is easier B1 B1.

Exam-style 3.4 ▪ 2 marks

For each, state the computational thinking skill used: **(a)** a designer keeps only the data the system needs and ignores the rest; **(b)** a designer notices the same calculation is needed in five places and writes it once as a function.

Solution 3.4

Worked steps

(a) **Abstraction** B1. (b) **Pattern recognition** B1.

Exam-style 3.5 ▪ 4 marks

A shop's stock-control program is designed using **decomposition**. One module, `Sales()`, records a sale and updates the stock level.

(a) **Explain** what is meant by decomposition, and **give** two benefits of designing this program that way.

(b) **Complete** the identifier table for `Sales()`.

identifier	data type	description
`ItemCode`	_____	_____
`QuantitySold`	_____	_____
`UnitPrice`	_____	_____

Exam-style 3.5 ▪ 4 marks

(c) **State** the purpose of an identifier table at the **design** stage, and **explain** why it is written before any code.
(d) The shop manager also wants a weekly report of items that fell below their reorder level. **Outline**, using **stepwise refinement**, the steps of a module `LowStockReport()`. Do **not** write pseudocode.
(e) **Explain** why `Sales()` and `LowStockReport()` should be separate modules rather than one long block of code.

Solution 3.5

Method: Decomposition

Worked steps

(a) Decomposition means breaking a problem into smaller sub-problems, each solved by its own module B1 B1. Benefits, any two: each module can be written and tested independently, so faults are easier to find B1; modules can be reused elsewhere, and several people can work on different modules at once B1.
(b) `ItemCode` — `STRING`, the unique code identifying the product B1. `QuantitySold` — `INTEGER`, the number of units in this sale B1. `UnitPrice` — `REAL` (or `CURRENCY`), the price of one unit B1. `InStock` — `INTEGER`, the number of units remaining after the sale B1.

Solution 3.5

(c) It lists every variable, its data type and what it is for B1, so the programmer knows exactly what to declare and nobody invents a second name for the same thing B1. Writing it first forces the data to be thought through before the logic, which is where most design faults are cheapest to fix B1.
(d) Any sensible four steps, in order: read the reorder level and the stock level for each item in turn B1; compare the two and select the items whose stock is below the level B1; accumulate or format those items into a report, including code, description and quantity remaining B1; output the report and record the date it was produced B1.
(e) Each does a **different job**, so keeping them separate means one can be changed or tested without disturbing the other B1; it also allows the low-stock check to be called from more than one place, such as at the end of a sale and again weekly B1.